Rapport TP 1

Diviser pour Régner

Introduction

L'objectif de ce TP, est de prendre en main le paradigme « diviser pour régner » et l'utiliser dans le contexte de réduction de complexité d'un algorithme. Dans ce TP nous allons nous concentrer sur la problématique du rectangle à superficie maximum (voir la figure 1), et ainsi développer des solutions de plus en plus pertinentes en termes de complexité algorithmique.

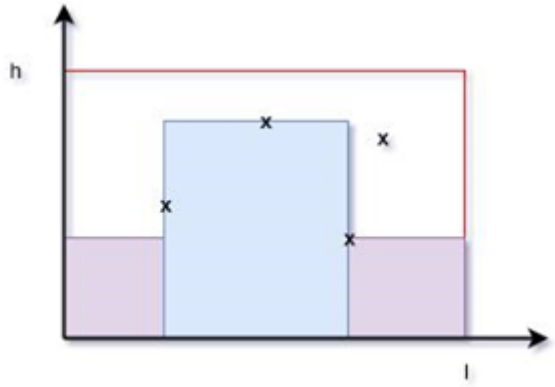

Figure 1 Illustration de la problématique du rectangle à superficie maximum, dont la base est sur l'axe des x, et qui ne contient aucun des n points définis.

Données de test

Nous nous sommes procuré des données plus tester nos solutions en termes de temps d'exécution, et ceci afin de confirmer nos hypothèses sur la complexité des algorithmes développés.

Q1. Méthode naïve : Complexité $O(n^3)$

Cette méthode part du postulat que le rectangle qu'on cherche a comme base l'axe des x (voir l'énoncé du TP), et que les sommets du rectangle sont délimités forcement par un point parmi les n, ou bien l'un des points spéciaux 0 et h pour l'axe des y, 0 et l pour l'axe des x qui représente les limites de notre domaine de travail.

Un rectangle de surface maximale respectant les contraintes a nécessairement deux sommets de la forme $(x_i, 0)$, $(x_j, 0)$, pourquoi ?

La base du rectangle est sur l'axe des x, il y a donc exactement deux sommets du rectangle avec y=0, ce qui explique que les deux points $(x_i, 0)$, $(x_j, 0)$ sont des sommets du rectangle final.

Comment exprimer la surface du rectangle de surface maximale respectant les contraintes et dont deux sommets sont $(x_i, 0)$, $(x_j, 0)$?

Soit $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ l'ensemble des n points ordonnées par abscisses.

La formule du rectangle qui possède la plus grande surface, qui a comme base l'axe des x, et qui ne contient aucun des n points est la suivante.

$$\max_{\forall\, i\in\{1,\dots,n-1\},\forall\, j\in\{i+1,\dots,n\}}(xj - xi) * \min(\{y_{i+1}, \dots, y_j\})$$

Pouvez-vous en déduire un algorithme en *O(n^3)*?

L'idée est de chercher un point pour la limite à gauche du rectangle, un autre en haut, et un dernier à droite. Ceci peut être fait avec 3 boucles **imbriquées** sur les points en entrée et aura donc une complexité de *O(n^3)* (*voir Methode01.py et Methode02.py). En effet, chaque boucle parcourt le tableau des points pour trouver les trois points cités ci-dessus. En majorant et dans le pire des cas, on pourra dire que chaque boucle parcourt tout le tableau, en supposons que la taille du tableau est « n » et sachant que les trois boucles sont imbriquées, les deux méthodes (Methode01,py et Methode02.py) s'exécutent en temps O(n^3).*

En O(n^2)?

Pour passer En O(n^2), il suffit de fixer une des trois valeurs qu'on est entrain de chercher, si on se fixe la hauteur à la hauteur minimum (ce qui est fait dans Q3)

Q2. Méthode Diviser pour Régner

L'algorithme décrit dans le fichier "divideandconquer.py", est un algorithme récursif. Dans le pire des cas, il y a autant d'appel récursif qu'il y a de points entre (0,0) et (0,l) (on remarque qu'on peut ne pas prendre en compte les appels dont les arguments satisfont la condition d'arrêt de la récursivité). La fonction principale "divide" fait appel à la fonction "min" qui s'exécute en temps linéaire par rapport à la taille de l'entrée. On peut voir cette récursivité sous forme d'arbre dont la racine est le premier appel vers la fonction "divide" et dont les nœuds sont les appels fils. Concernant les nœuds de l'arbre, le tableau passé en entrée à la fonction "min" pour chaque appel fils i a comme taille n/2^i de l'appel père. Les appels se trouvant sur les feuilles de l'arbre, quant à eux, n'utilisent pas la fonction "min" puisqu'ils tombent sur la condition d'arrêt de la récursivité. Ce qui nous amène à dire que la fonction "divide" se calcule en O(n log n) où n est la taille du tableau initial.

Q3. Hauteur limitée

En ayant un apriori sur la hauteur limite des points, on peut passer d'un algorithme d'une complexité O(n3) à un algorithme d'une complexité O(n2). La méthode consiste à chercher seulement la limite à gauche et à droite du rectangle, et de fixer la hauteur de celui-ci à la hauteur minimum sur tous les points parcourus entre xi et xj

Vue que l'algorithme ne contient que deux boucles imbriquées sur les données de départs, sa complexité est donc de O(n2). En effet, le fait de fixer la valeur de la hauteur à la valeur minimum, réduit considérablement le nombre de parcours à faire sur le tableau et réduit, de ce fait, le degré du polynôme décrivant la complexité du traitement, puisque la hauteur à été fixée, plus besoin d'une boucle pour chercher la hauteur, il reste donc que deux boucles. En supposant que chaque boucle parcourt entièrement le tableau de points en entrée, ce qui est le pire des cas (voir On2,py), le programme va parcourir donc n fois un tableau de taille n, ce qui fais une complexité de O(n2).